DCI PLC

a computer-simulated PLC

By I. de Coninck

PLCs are microcontroller circuit boards that are frequently used in a wide variety of industrial control and automation applications. The PLC described here is implemented in software. A complete PLC is simulated on a standard Windows PC. Using an external connection box, the PC can also sense and control many kinds of real items.

The abbreviation 'PLC' stands for 'Programmable Logic Controller'. These devices are also sometimes referred to as 'freely programmable controllers' in the trade. This term refers to an electronic circuit that can be used to automate machinery and processes. A PLC thus can replace a conventional relay logic circuit.

A PLC always consists of a CPU module and several digital input and output modules. The number of input and output modules depends on the size of the process to be controlled.

One of the major advantages of a PLC is that it makes it very easy to extend or modify the process. Consequently, PLCs have for years been an indispensable element in the automation of industrial systems.

Recently, PLCs have also started to appear in domestic applications. After all, people would also like to have certain things automated in their houses. Automatic garden sprinkler systems, awning controllers, security systems and many other types of systems can easily be controlled using a PLC. Extensions or modifications can be made relatively easily after the system has been installed.

Figure 1 shows an example of a simple motor circuit, in one case built using pushbuttons and relays and in the other case using a PLC. If you want to modify the operation of the circuit, you will have to modify the hardware if you use the relay circuit, but with the PLC all you have to do is to make a few changes to the program.

The objective of the DCI PLC design is to implement a PLC in a relatively simple and inexpensive manner using a minimum of hardware, a serial interface cable and PC software. This saves the trouble and expense of building a real PLC. In addition, programming this controller is very intuitive and user-friendly.

The concept

DCI PLC is actually a program that replaces the CPU module of a PLC, and within which control programs can be developed and modified. With the aid of a serial interface cable and some hardware, the states of 32 inputs can be monitored and 32 outputs can be driven. There are also 32 timers, 32 counters and 32 flags.

The program runs under the Windows 95 and 98 operating systems. The program itself will also work under Windows NT, but the hardware cannot be connected because NT does allow direct hardware addressing.

The minimum system requirement is a 100-MHz Pentium processor. The installed software takes up approximately 3 MB, including the examples.

The hardware of this design consists of nothing more than an assembly that converts PC serial port information into a number of outputs and inputs. The actual input and output interface must be developed by the user. It can be connected to the DCI PLC interface circuit board using flat cables. It is not possible to design a universal interface that satisfies all possible needs, but most *Elektor Electronics* readers should not find it all that difficult to develop an interface for a specific application. In addition, at the end of this article we suggest some sample interfaces.

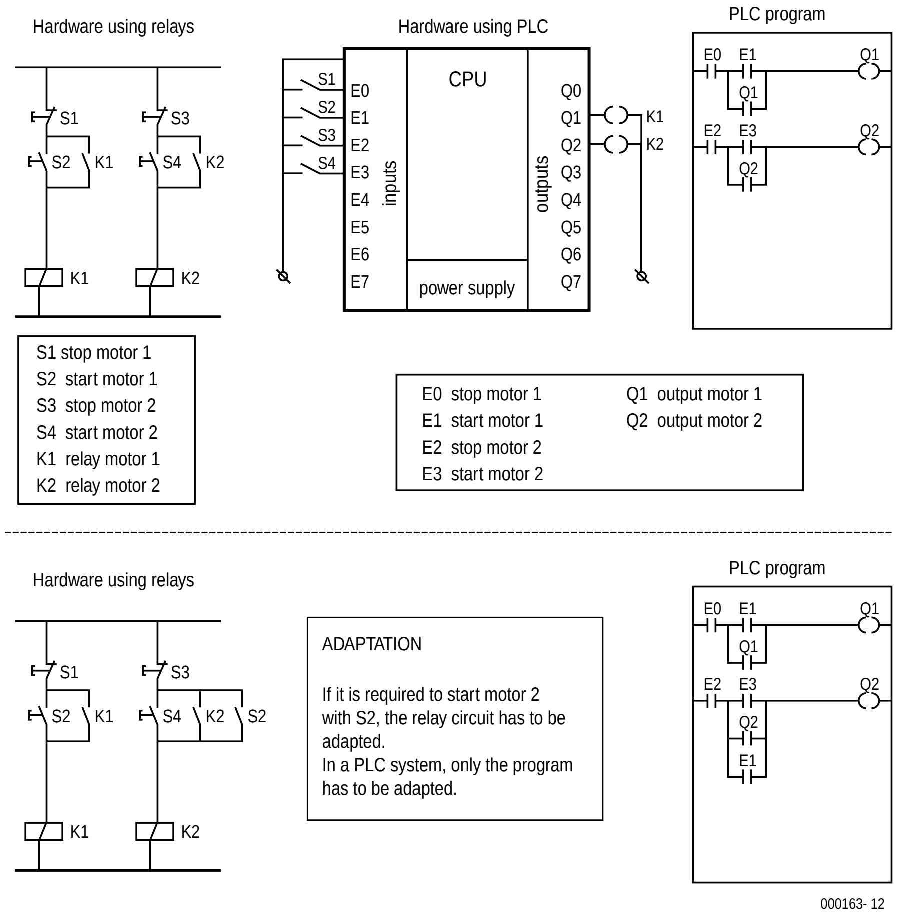


Figure 1. Example of a start/stop circuit for two motors, in one case using relays and in the other case using a PLC.

The hardware

If you look at the schematic diagram for the hardware (**Figure 2**), you will see that it actually consists of little more than a handful of CD4094 and CD4021 shift registers.

The 4094 shift register is an integrated circuit containing eight flip-flops connected in series. With each clock signal, the logic level on the input ('0' or '1') is shifted by one position, so that after eight clock pulses and one strobe pulse we have an 8-bit output signal available. In our circuit, four of these shift registers are connected in series (IC5–IC8). This gives us a total of 32 outputs.

The 4021 shift register has the advantage (relative to the 4094) that can also be used 'the other way around'. It can thus be used to read in the levels on its eight inputs. The output levels of the eight internal flip-flops appear one after the other on the serial output when clock pulses are applied to the clock input of the IC.

We have chosen to use the PC serial port for the connection to the hardware because it offers several advantages, namely:

- The serial port is relatively robust (short-circuit protection).
- The cable may be connected while the PC is operating.

In the interface hardware, the RS232 signal levels are limited by Zener diodes D1-D3 to +4.7 V and –0.6 V. Resistors R1-R3 provide current limiting.

For more information about using the PC serial interface, see the series 'PC Serial Peripheral Design' (starting with the September 2000 issue of *Elektor Electronics*).

In addition to the shift registers, the circuit board includes a voltage regulator to provide a 5-V supply voltage. The input voltage may range from 10 V to 20 V. There are also 32 LEDs (D5-D36) with associated series resistors, which make the levels on the 32 outputs visible. Finally, we can note that all 32 inputs are fitted with pull-up resistors, so that a High level is present on any unused input.

Figure 3 shows the layout of the printed circuit board that has been designed for this hardware. It is a double-sided board, in order to avoid the use of wire bridges. Assembling the circuit board is cer-

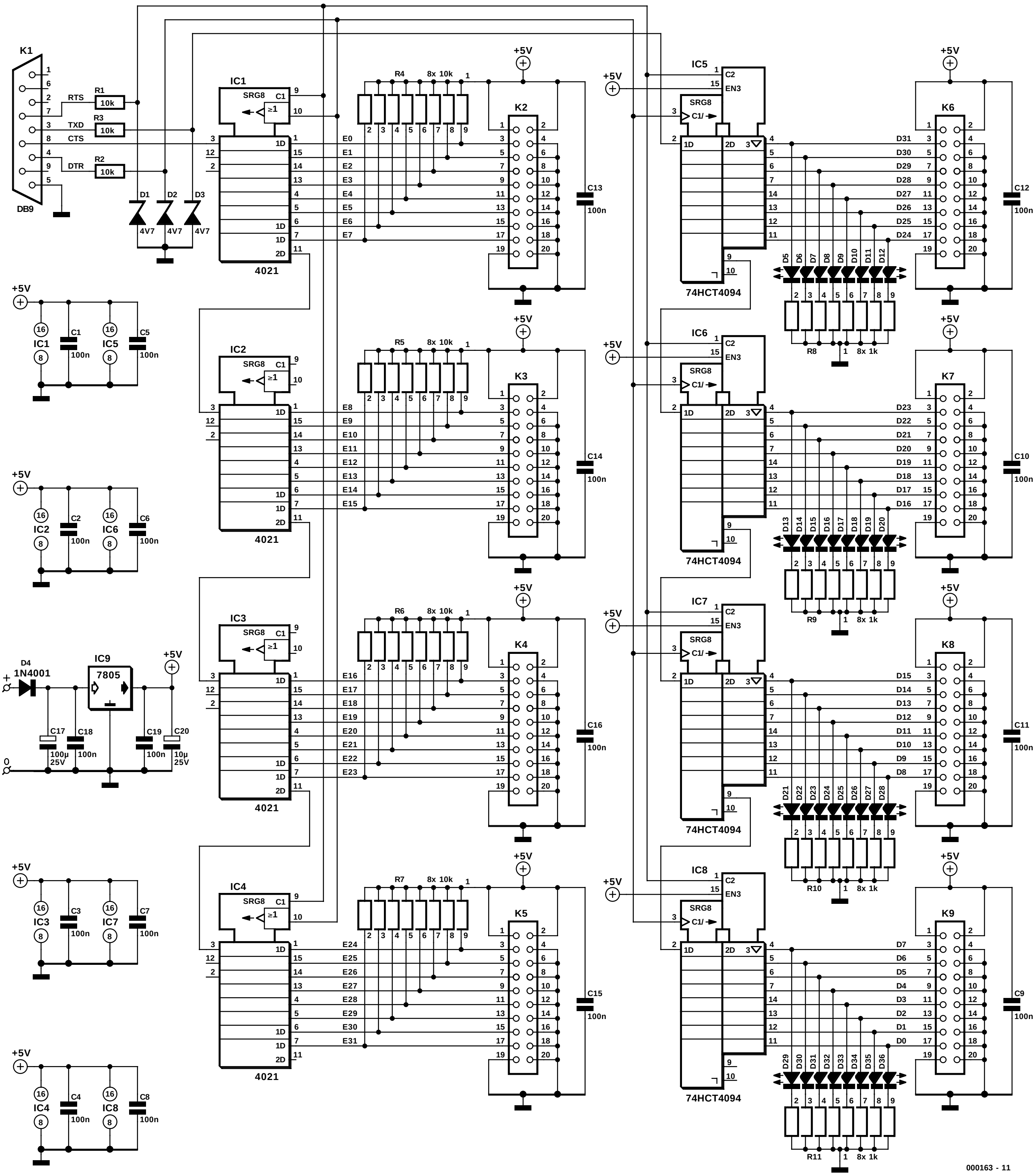


Figure 2. The hardware primarily consists of a handful of 4021 and 4094 shift registers.

tainly not difficult. Since the ICs draw only a small amount of current, the 7805 does not need a heat sink.

We have chosen to use rectangular LEDs, so that they can be placed close together in groups of eight located exactly opposite the associated header pins.

Standard 2×10 pin headers or box headers are used for connections to the outside world. A length of flat cable can easily be connected to each header using a press-on (IDC) connector. Each connector has +5 V and ground on the two ‘extra’ pins, and a ground connection is present opposite each

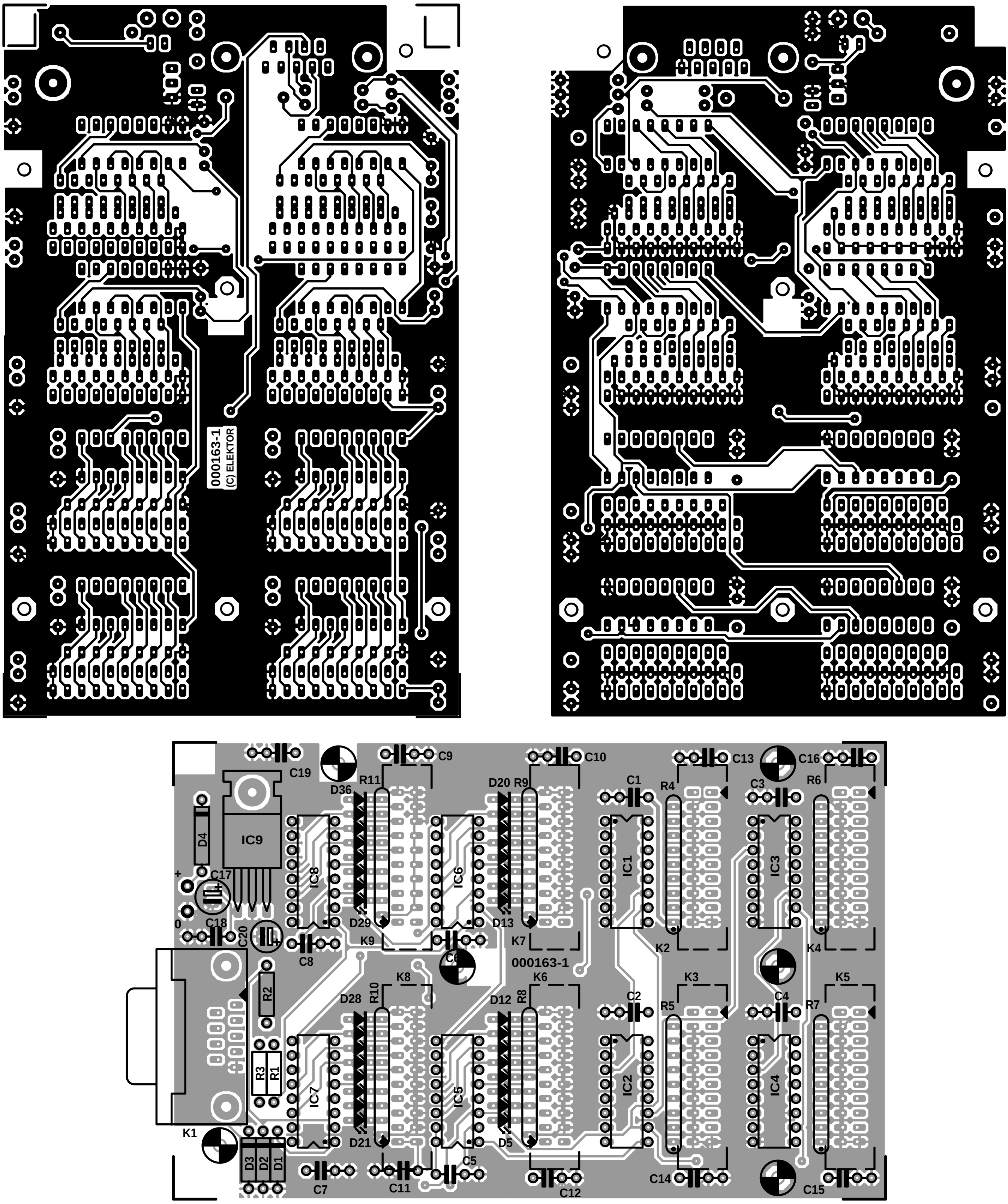


Figure 3. The double-sided printed circuit board for the DCI PLC.

COMPONENTS LIST

Resistors:
R1-R7 = 10kΩ
R4-R7 = SIL array 8 x 10kΩ
R8-R11 = SIL array 8 x 1kΩ

Capacitors:
C1-C16, C18,C19 = 100nF
C17 = 100µF 25V radial
C20 = 10µF 25V radial

Semiconductors:
D1,D2,D3 = zener diode 4.7 V 400mW
D4 = 1N4001
D5-D36 = LED, rectangular, red, high efficiency
IC1-IC4 = 4021
IC5-IC8 = 74HCT4094
IC9 = 7805

Miscellaneous:
K1 = 9-way sub-D socket (female), PCB mount
K2-K9 = 20-way boxheader
Case: Teko 160x95x45 mm (Conrad Electronics #526177)
PCB, order code **000163-1** (see Readers Services page)
Disk, Windows program, order code **000163-11**

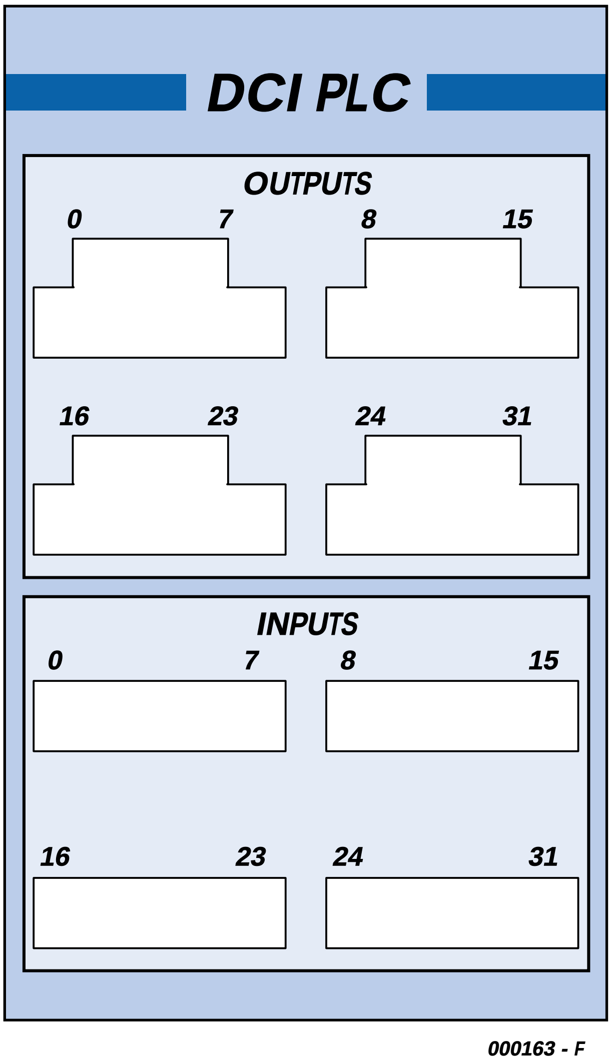


Figure 4. Example of a matching front panel foil for the circuit board shown in Figure 3.

signal connection (see the schematic diagram). When assembling the circuit board, keep in mind that he connectors must be fitted on the solder side. This will make the numbering of the inputs and outputs match the numbering on the front-panel foil (see **Figure 4**).
If you want to finish the assembly a tidy manner, you can mount the circuit board in a small enclosure fitted with a front panel foil, such as the on shown in Figure 4.

Using the software

The DCI PLC program is designed to allow you to easily prepare a PLC program by simply placing blocks.
The length of the ladder logic diagram that is constructed in this manner is limited to 99 lines. Ladder logic diagrams, which are in many ways similar to relay logic diagrams, are frequently used to graphically generate programs. When putting together a ladder logic diagram, you should always bear in mind that the symbols in the diagram represent instructions rather than physical contacts.
When the instructions in the program are executed, the states of the inputs are monitored. Depending on these states, various outputs are set either High or Low.

The instruction set

The instruction set of the DCI PLC consists of the following elements (see Figure 5):

1. **Test for ON**. This instruction checks whether a voltage is present. It can be assigned to an input, output, flag, timer or counter. The status of this instruction is TRUE whenever a voltage is present (if the instruction is assigned to an input or output).

2. **Test for OFF**. This instruction checks whether a voltage is absent. It can be assigned to an input, output, flag, timer or counter. The status of this instruction is TRUE whenever no voltage is present (if the instruction is assigned to an input or output).

3. **Positive edge detection**. The status of this instruction becomes TRUE when a rising edge is detected. This remains valid for the duration of only one program cycle. This instruction can only be assigned to a flag.

4. **Negative edge detection**. The status of this instruction

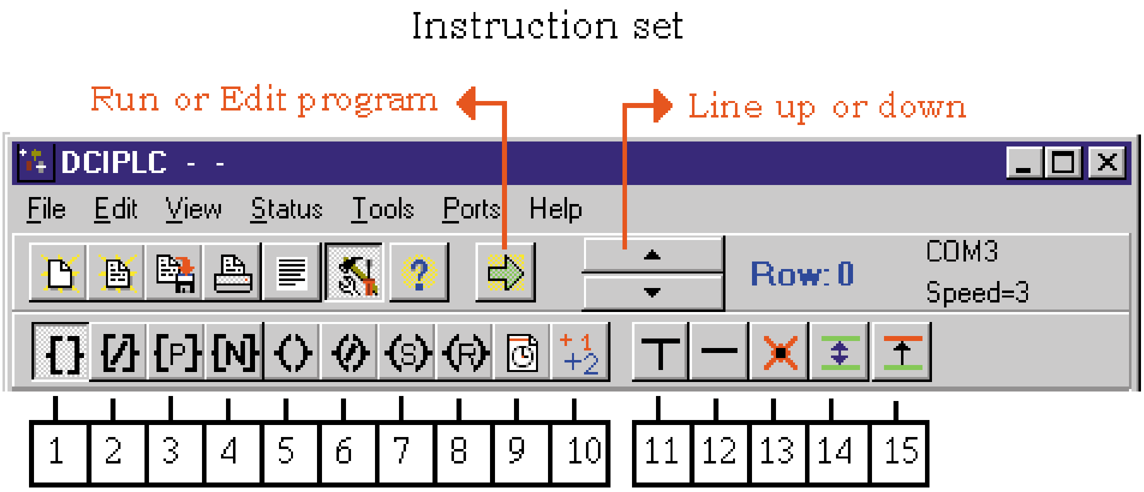


Figure 5. The toolbars of the DCI PLC program. The numbers under the buttons refer to the descriptions of these buttons in the text.

becomes TRUE when a falling edge is detected. This remains valid for the duration of only one program cycle. This instruction can only be assigned to a flag.

5. This symbol stands for an **internal or external output** and can be assigned to a flag or an output. This output will become High when the result of the logical evaluation of the entire row is TRUE.

6. This symbol stands for an **inverted internal or external output** and can be assigned to a flag or an output. This output will become High when the result of the logical evaluation of the entire row is FALSE.

7. **SET internal or external output**. This symbol can be assigned to a flag or an output. This output will become High when the result of the logical evaluation of the entire row is TRUE. Even if the logical evaluation of the entire row later becomes FALSE, the output will remain High. Only **RESET output** can change the output state back to Low.

8. **RESET internal or external output**. This symbol can be assigned to a flag or an output. This output will become Low when the result of the logical evaluation of the entire row is TRUE. Even if the logical evaluation of the entire row later becomes FALSE, the output will remain Low. Only *SET* **output** can change the output state back to High.

9. The **timer** is an instruction that remains High or Low for a set interval (it can also be alternately High and Low). This instruction must always be placed in the last column of the row.
   The timer starts when the logical evaluation of the entire row is TRUE, and it continues to run as long as the logical evaluation remains TRUE. If the logical evaluation of the entire row becomes FALSE, the timer is reset. The variable *t1* represents the interval in seconds during which the timer is Low, while the variable *t2* represents the interval in seconds during which the timer is High. The variables *t1* and *t2* may have values from 1 to 9999. When a timer is assigned, a value must be assigned to *t1* or *t2*, since otherwise an error message will occur. A 'flasher' can be created by assigning values to both *t1* and *t2*. To implement very long intervals, a timer can be combined with a counter.

10. The **counter** instruction generates a High level after a defined number of pulses. Counting starts when the logical evaluation of the entire row is TRUE, and it continues as long as the logical evaluation remains TRUE. If the logical evaluation of the entire row becomes FALSE, the counter is reset. The counter value can be preset from 1 to 9999. A counter has two inputs: one for counting up and the other for counting down. A flag, input, timer, output or counter can be assigned to each of these inputs. At least one of the inputs must receive an assignment. The state of the counter becomes High only when the count equals the preset value!
    The counter instruction must always be placed in the final column.

11. Start or end of an *OR* **instruction**. This instruction can only be placed in columns 1 through 8, and it must always have a start and an end. Only one OR instruction can be used per row, and the maximum span of an OR instruction is 7 rows. In the `/examples` folder on the diskette, there are two files (`or.plc` and `bador.plc`) that show what can be done (and what cannot be done) with an OR function.

12. **Line** instruction. This instruction is the equivalent of a signal line in a schematic diagram. It always shows the level of the previous cell.

The following tools are available for making programs:

`delete an instruction`
`insert a row`
`delete a row`

Note: A 'flag' is a sort of internal output that

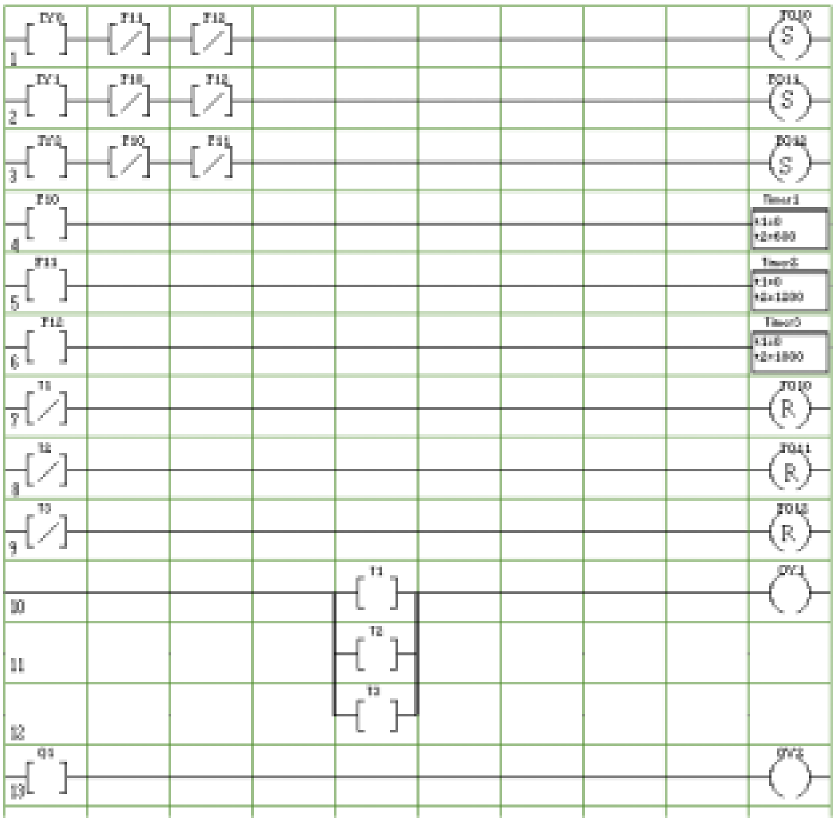
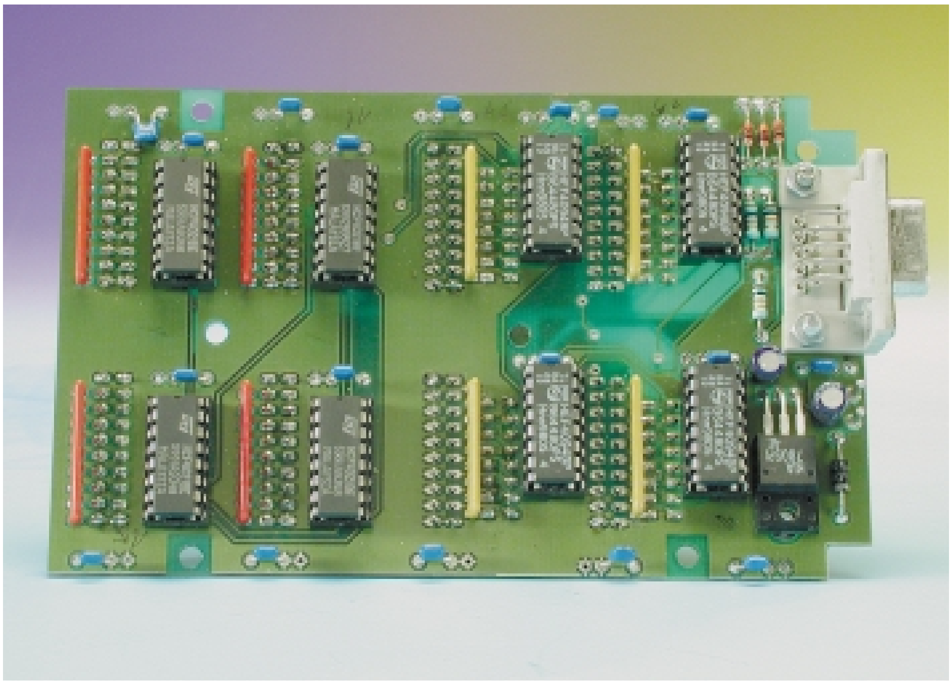


Figure 6. Structure of the sprinkler program.

is used to store an intermediate result. A flag can be 'set' and 'reset' just like an output, so it is effectively an invisible output.

The program also runs under Windows ME.



Creating a program

When generating a program, it is a good idea to work from left to right and to only start a new row after the current row has been completed. The second point is also necessary because the program checks for empty rows.

You should also make a habit of always assigning a particular output only once when generating a PLC program.

Every PLC works in the following manner:

1. When the PLC starts up (changes to Run mode), all outputs are set to inactive (no voltage).
2. The states of all inputs are read in.
3. The user program processes the instructions sequentially using the signal states acquired in Step 2.

If the state of an input signal changes while the instructions are being processed, this does not have any effect during the cycle currently being executed. However, the states of timers, counters and flags are taken into account during the processing of the program.

4. After the user program has been processed, the results are sent to the outputs.
5. Steps 2, 3, and 4 are continuously repeated in a cyclic manner.

If the same output is used more than once in a program, this output will be assigned the state that it receives in the last row in which it is located!

The DCI PLC program processes a complete user program 20 to 100 times per second. The actual rate depends on the speed of the PC and the length of the program.

An example

The following example is a simple sprinkler controller for a garden.

We assume that two locations in the garden must be watered. We want to be able to choose from three options, depending on how dry the ground is: a short sprinkling time, a long sprinkling time and a very long sprinkling time.

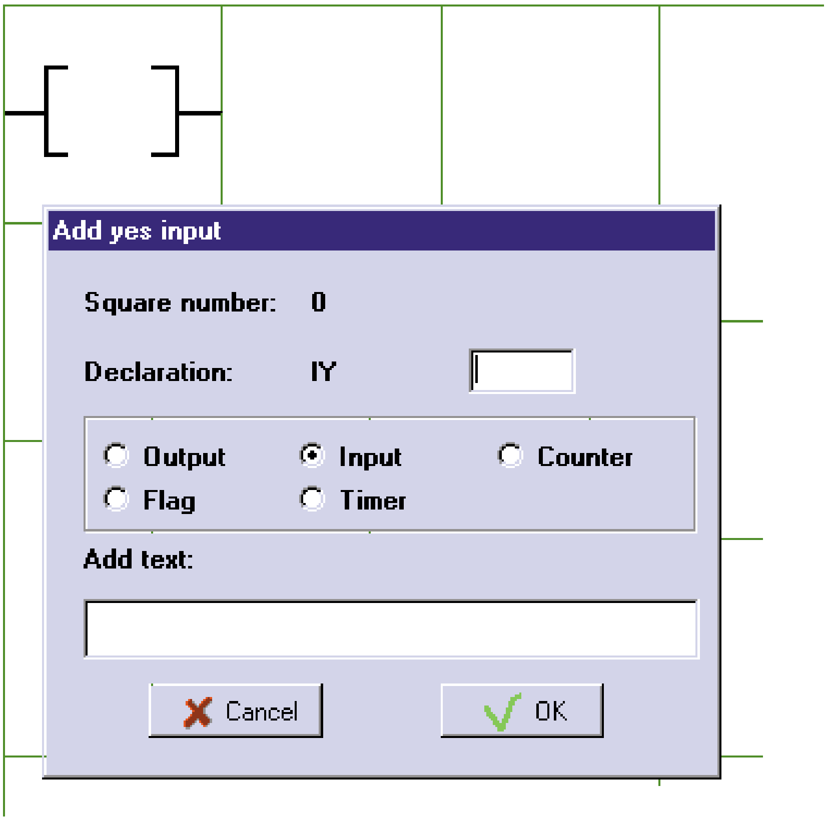
This is what we need:

Two sprinkler valves:

QY1: Output Yes no. 1
QY2: Output Yes no. 2

Three selection pushbuttons:

IY0: Input Yes no. 0,
for 10 minutes' sprinkling



Square number: 0

Declaration: IY

○ Output ● Input ○ Counter

○ Flag ○ Timer

Add text:

Figure 7. When an instruction is placed, a dialogue window appears for the assignment, optional comment and so on.

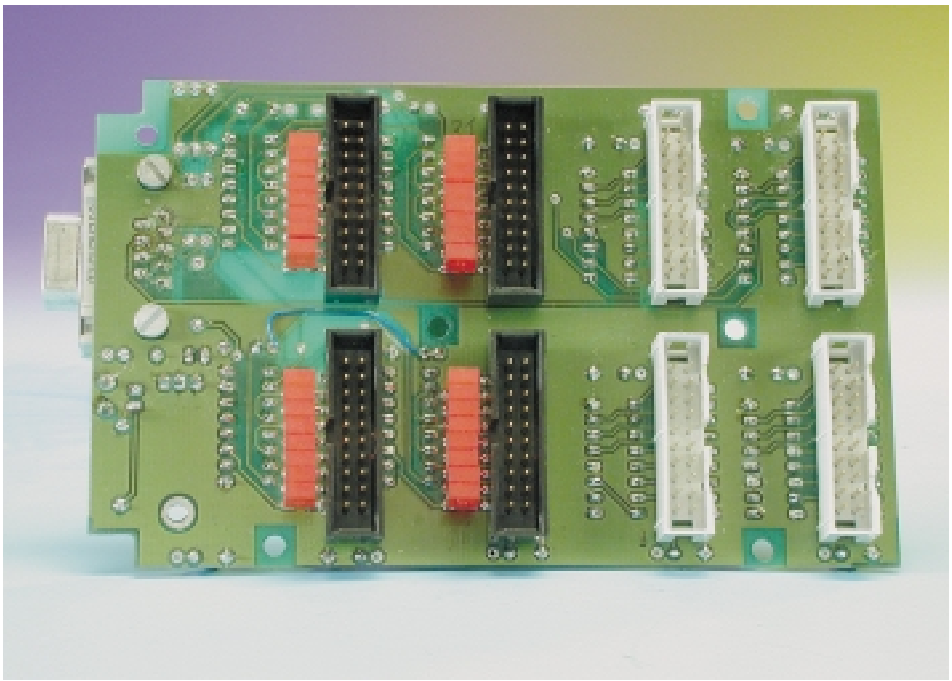
IY1: Input Yes no. 1,
for 20 minutes' sprinkling
IY2: Input Yes no. 2,
for 30 minutes' sprinkling

Now let's see how to enter the program.
First, in the DCI PLC program, select the button **Test for ON** in the toolbar (number 1 in Figure 5) and click on cell 0 in row 1.
The chosen instruction will now be located in this cell, and a dialogue window will appear to allow you to assign the input and add a comment. In this dialogue, select 'Input' and enter '0' as the assignment. This will cause input 0 of the PLC to be assigned to this instruction. If you wish, you can enter a comment, which can later be viewed by clicking on the cell with the right-hand mouse button.
If you now click on 'OK', input 0 will be assigned. This is shown by the fact that 'IY0' appears at the top of the cell.
Next, place a **Test for OUT** instruction (2) in each of the following two cells, and assign these instructions to F11 and F12.

F11 and F12 are flags that will be set and reset later on in the program. They are used to prevent another time from being chosen after one of the pushbuttons has been pressed.
Now select the **Line** instruction (12) and click once in cells 3 through 8. The lines have now been placed.
Select the **Set output** instruction (7) and click in cell 9. The output instruction will be placed and a dialogue window for assigning an output will appear. Select 'Flag' and enter '10'. You can also add a comment if you wish.
Now you can see that the first row is already finished. Generate the remaining rows in a similar manner, as shown in **Figure 6**.
Once you have finished, you can save the program, and it is then ready to be tested.

To test the program, or to run it without the hardware, you can use the simulation mode. To do so, click on 'Simulation mode' in the 'Status' menu. A window will appear, in which you can enter the states of the 32 inputs and check the states of the outputs.
Set input 1 High and then Low. This starts timer 2, and outputs 1 and 2 will then remain High for 20 minutes.
The program gives a visual representation of the states of the inputs and outputs and also displays the current values of the timers.
Once sprinkling has started, setting one of the inputs High no longer has any effect. This is because rows 1, 2 and 3 have been blocked.

After the hardware has been connected, you



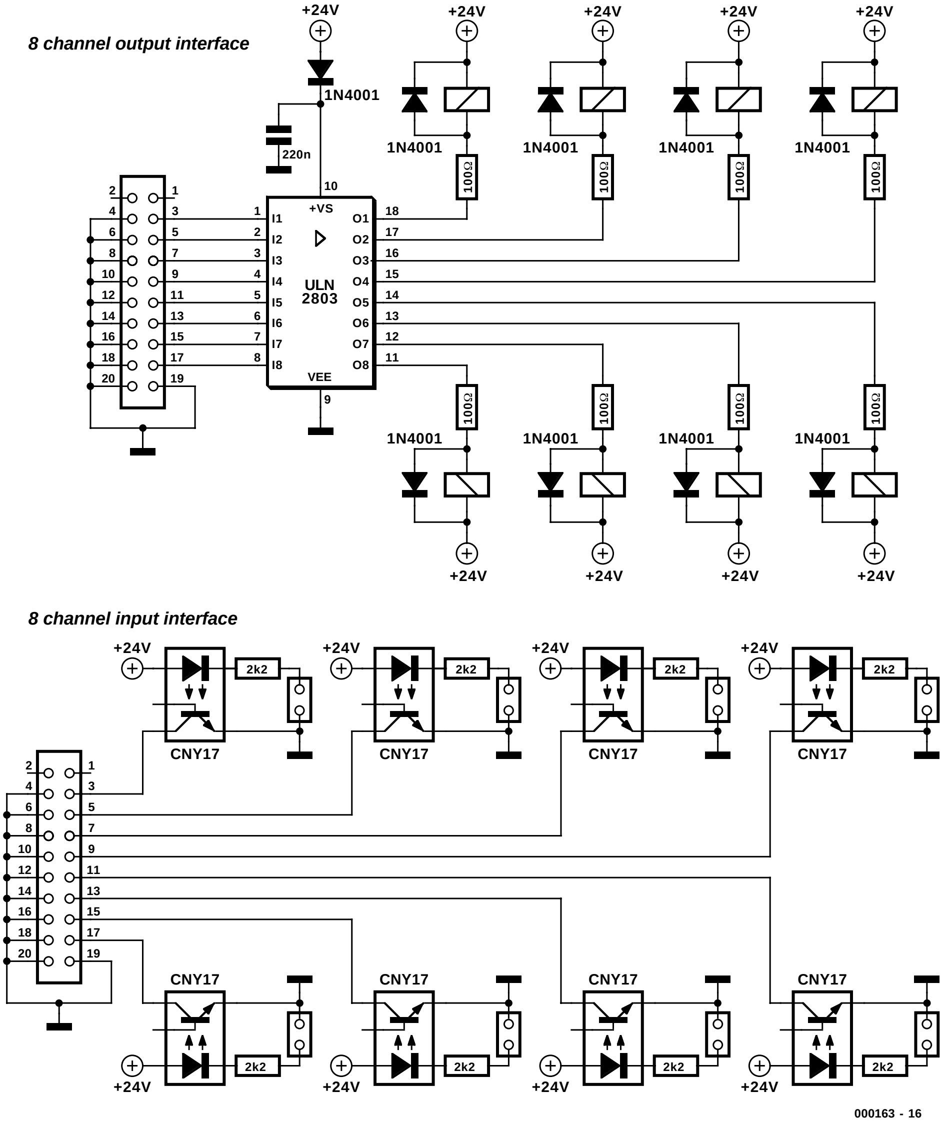


Figure 8. Two examples of possible interfaces: (a) eight relay-driven outputs; (b) eight opto-coupler inputs.

can make the proper port settings using the ‘Ports’ menu.

You can choose from COM1–COM4, and you can also set the speed. It is a good idea to initially set a low speed and then check in RUN mode whether the inputs generate the proper responses and the outputs are correctly driven. Click on ‘Run mode’ in the ‘Status’ menu, or click on the toolbar button with the green arrow.

You can return to the edit mode at any time by clicking on ‘Edit mode’ in the ‘Status’ menu or by clicking on the button with the red circle. This causes all hardware outputs to be reset to zero.

The following tips will help you obtain the best results from the program in the Run mode:

– Make sure that as few programs as possible are active.
– Disable screen savers.
– Do not start any other programs.
– Do not install a diskette or CD-ROM in the PC while the program is running.

These measures allow Windows to reserve the maximum possible amount of time for the DCI PLC program, which has a beneficial effect on its performance. If a CD-ROM is inserted in the tray of the CD player, for example, the DCI PLC program will stop responding for a few seconds, since Windows gives priority to the CD-ROM drive. During this interval, the timers are no longer updated and inputs are not checked. If you observe the above recommendations, the program will check the inputs at least ten times per second and the timer error will not exceed two seconds in 24 hours.

On a 400-MHz Pentium machine, the sprinkler program achieves a speed of 90 cycles per second.

The diskette containing the program (which is also available from the *Elektor Electronics* website) also includes some other simple examples that you can use for practice.

Interfacing

For coupling the controller to various types of devices, relays and sensors, readers of *Elektor Electronics* can draw on a vast store of circuits that have been published in earlier issues. Still, we would like to provide a pair of simple examples.

Figure 8a shows an 8-channel output interface consisting of a buffer IC and eight relays. This can be used to supply power to eight devices or circuits (with the proper choice of relay, it is even possible to switch the mains voltage, but you must make sure to use a safe construction).

Figure 8b shows an 8-channel input interface. This consists of eight opto-couplers with associated current-limiting resistors. If the two terminals are connected together, a High level appears on the corresponding PLC input.

(000163-1)

The author has his own website for the DCI PLC:

http://home.planetinternet.be/~dc11cd/